

4. Défilement et mise à l'échelle

Défilement (Scrolling)

Dans une application, il peut être nécessaire de savoir positionner son dessin par rapport à sa zone d'affichage ou de le mettre à l'échelle, par exemple lors d'une impression, que nous ne traiterons par ici.

Ceci est aussi vrai pour l'affichage à l'écran, et on le traite généralement en permettant de déplacer la zone de dessin par rapport à la zone d'affichage, ou de zoomer sur ce même dessin.

On peut imaginer, comme le montre la figure ci-contre, que notre application est constituée de deux espaces plans, l'un qui contient les objets à dessiner (le **Dessin**, cadre bleu), l'autre qui représente l'écran de visualisation (l'**Ecran**, cadre rouge). Jusqu'ici, ces deux espaces étaient parfaitement confondus.

Lorsqu'on utilise un ascenseur (*scrollbar*), cela revient à faire glisser (en géométrie, on parle de *translation*) les deux espaces l'un par rapport à l'autre. Si l'on admet que l'**Ecran** reste fixe (notre écran ne bouge pas), cela revient à dire que c'est le **Dessin** que l'on a fait bouger par rapport à l'**Ecran** (mais les objets sont restés immobiles à l'intérieur du **Dessin**).

Pour afficher à l'**Ecran** un point du **Dessin** (l'ellipse bleue sur notre dessin), il faut donc calculer ses coordonnées **Ecran** en fonction de ses coordonnées dans le **Dessin**. Ceci ne dépend clairement que des positions relatives des coins supérieurs gauches des deux rectangles.

Le coin gauche de l'**Ecran** étant conventionnellement fixé à (0, 0), appelons **Origine** le coin supérieur gauche du **Dessin**. Il est bien clair que si (x, y) sont les coordonnées du point dans le **Dessin**, ses coordonnées à l'**Ecran** sont (X', Y') avec :

$$X' = x + \text{Origine.X}$$

$$Y' = y + \text{Origine.Y}$$

Remarque : ce point **Origine**, ainsi que des fonctions de conversion **Dessin - Ecran** décrites juste après, vont être utilisés à plusieurs endroits dans le code (Feuille de dessin et méthode de dessin des nœuds). La propriété **Origine** sera donc définie comme propriété statique dans la classe **Outils** que nous avons déjà créée, la rendant ainsi accessible dans toutes les classes du projet.

```
static public Point Origine { get; set; } = new Point(0, 0);
```

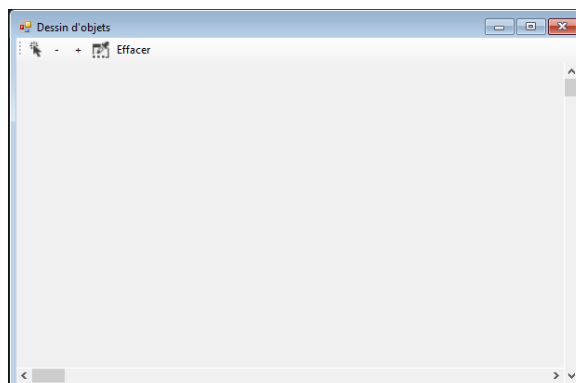
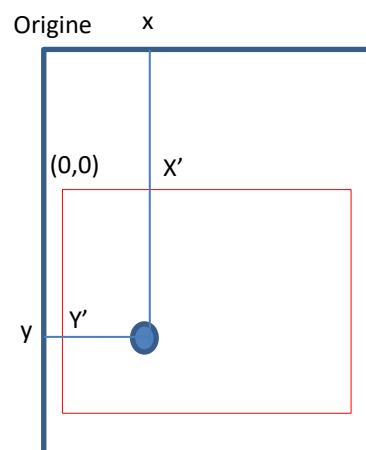
L'utilisation des ascenseurs va donc simplement devoir modifier les coordonnées du point **Origine** par rapport au point (0, 0), ces coordonnées deviendront ≤ 0 .

Par glisser-déplacer, on peut ajouter à la feuille un *ascenseur* et un *translateur*, à savoir un **VScrollBar** et un **HScrollBar** que l'on collera à droite et en bas de la feuille respectivement, grâce à leur propriété **Dock**.

Pour les adapter au dessin en cours on pourra modifier quelques-uns de leurs paramètres de base :

```
vScroll.Minimum = 0;  
vScroll.Maximum = 800;  
vScroll.SmallChange = 4;  
vScroll.LargeChange = 50;
```

Les deux premiers précisent l'intervalle de valeurs de l'ascenseur, les autres les valeurs dont il est incrémenté ou décrémenté à chaque fois que l'on clique sur les extrémités (**SmallChange**) ou sur l'ascenseur lui-même (**LargeChange**).



Un seul de leurs événements nous sera utile, l'évènement **Scroll**, le code pourra ressembler à :

```
private void VScroll_Scroll(object sender, ScrollEventArgs e)
{
    Outils.Origine = new Point(Outils.Origine.X, -e.NewValue);
    Refresh();
}
```

Nous l'avons utilisé ici pour modifier celle de la propriété **Origine**, de type **Point**.

Reste maintenant à appliquer effectivement le changement au dessin (le « défilement »), *sans modifier les coordonnées des points* naturellement, uniquement celle à l'**Ecran**.

Il va donc être nécessaire, à plusieurs endroits dans le code (Feuille de dessin et méthode de dessin des nœuds), de réaliser des conversions entre les coordonnées **Ecran** et les coordonnées dans le **Dessin**. La classe statique **Outils** est un emplacement commode pour écrire le code de ces fonctions.

Exemple : voici celle de la transformation des coordonnées d'un point dans le **Dessin** vers ses coordonnées à l'**Ecran**.

```
public static Point DessinVersEcran(Point p)
{
    return new Point(p.X + Origine.X, p.Y + Origine.Y);
}
```

Exercice 29. Modifier le code de la classe **Outils**, écrire le code des méthodes **DessinVersEcran** (fournie ci-dessus) et **EcranVersDessin**, puis apporter les modifications nécessaires aux méthodes de dessin.

Testez ! Dessinez un petit graphe (nœuds et traits) et utilisez les ascenseurs : votre graphe doit s'afficher correctement, MAIS...

Nouveau problème : l'affichage fonctionne, mais les méthodes de sélection et de déplacement des objets ne fonctionnent plus !



Exercice 30. Remettre en état de marche le mécanisme de sélection et de déplacement en utilisant aux bons endroits les fonctions de conversion de coordonnées.

Complément 7. Rajouter une fonctionnalité permettant de déplacer le dessin simplement en le sélectionnant avec la souris.

Mise à l'échelle (zooming)

La mise à l'échelle du dessin est une seconde opération géométrique bien connue, une *homothétie*.

Elle est caractérisée par deux paramètres : son rapport (le facteur de *zoom* et son centre). Pour rester simple, nous allons ne considérer que des homothéties centrées au point (0, 0) et nous concentrer sur le facteur de zoom.

Pour contrôler la mise à l'échelle, on peut imaginer rajouter des boutons  et , mais le zoom peut aussi être contrôlé grâce à la molette de la souris dont le mouvement sera traité en associant une réponse à l'évènement **MouseWheel**. Malheureusement, cet évènement n'apparaît pas dans la liste des évènements de la souris, et il faut le traiter « à la main » :

```
MouseWheel += FeuilleDessin_MouseWheel;
```

Le code associé (pour l'augmentation du facteur de zoom) pourra ressembler à celui-ci (pour l'agrandissement), qui utilise une propriété de type **float** et de nom **Zoom** (cette même méthode peut aussi traiter le défilement de l'ascenseur) :

```
private void FeuilleDessin_MouseWheel(object sender, MouseEventArgs e)
{
    if (Control.ModifierKeys == Keys.Control)
    {
        if (e.Delta > 0) // agrandissement
        {
            Outils.Zoom *= 1.1f;
            if (Outils.Zoom > 15f)
                Outils.Zoom = 15f;
        }
        if (e.Delta < 0) // rétrécissement
        {
            // à compléter
        }
        Refresh();
    }
}
```

Ici donc, c'est l'appui de la touche « Ctrl » + la molette de la souris qui déclenche le zoom / dezoom du dessin.

Reste maintenant à appliquer effectivement le zoom comme nous l'avons fait pour le défilement : il suffit pour cela de modifier le code des méthodes de conversion des données **Dessin** en données **Ecran**.

(Un nouveau problème apparaît ici, celui de la mise à l'échelle des polices de caractères, si on désire afficher des textes avec les nœuds.)

Exercice 31. Implémenter la mise à l'échelle.

Exemple : voici celle de la transformation des coordonnées d'un point dans le **Dessin** vers ses coordonnées à l'**Ecran**. Et aussi avec une surcharge, pour la taille.

```
4 références
public static Point DessinVersEcran(Point p)
{
    return new Point((int)(p.X * Zoom) + Origine.X, (int)(p.Y * Zoom) + Origine.Y);
}
1 référence
public static Size DessinVersEcran(Size t)
{
    return new Size((int)(t.Width * Zoom), (int)(t.Height * Zoom));
}
```

Complément 8. (Très compliqué) Modifier le zoom pour qu'il s'effectue à partir du centre du dessin (homothétie par rapport au centre de la page).

Synthèse

Cette section a traité de quelques points important de l'*interface* comme le *défilement* et la *mise à l'échelle*, en séparant l'espace des données (le **Dessin**) de l'espace de représentation (l'**Ecran**).